MARKET RESEARCH BRIEF

The One-Year Gap

Nearly every company runs its business on software borrowed from other companies. Most of it is more than a year out of date — and nobody is fixing it.

What this is: an independent measurement of 32 widely used software vendors, covering how far behind their customers' systems have fallen.

Date: August 2026

Summary

> Modern companies do not build most of what their software does. They rent it. Payments come from one vendor, text messages from another, login from a third, maps from a fourth. Each of those connections has to be kept current. This research measured whether they are. They are not.

79%
of live traffic to the typical vendor comes from customers running an outdated version

376
days old, on average, is the version of vendor software actually running in production

23/32
vendors measured where most customers are behind, not a minority

Three things are true at once, and together they describe a gap in the market:

1. **Customers are far behind.** Across 32 major software vendors, the typical vendor sees roughly four out of five of its connections coming from customers running an old version. The median version in active use is over a year old.
2. **Vendors keep changing things.** These vendors regularly ship changes that stop old code from working. The heaviest offenders do it more than a dozen times a year.
3. **People are visibly stuck.** More than 1,200 public complaints were found from engineers whose systems broke, or who were trapped mid-upgrade. Nearly a third of those were still unresolved when this research was done.

The gap between what vendors ship and what customers run is where outages, security exposure, and wasted engineering time live. Nothing today closes it automatically.

The problem, in plain language

Think of a vendor as an electricity company and its product as a wall socket. Your business builds a plug that fits that socket. Everything works.

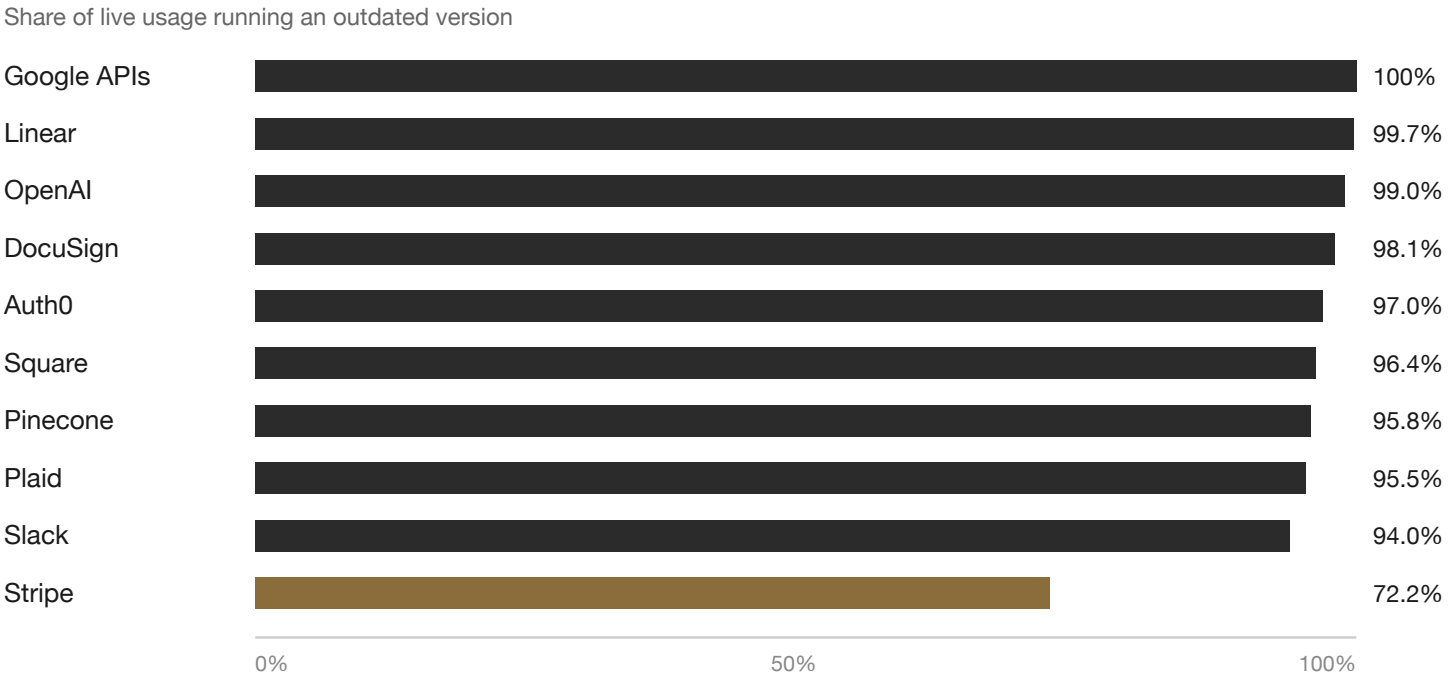
Then the electricity company redesigns the socket. They announce it — in a newsletter, on a blog, in a page of release notes nobody on your team has time to read. Your plug still fits, for now, because they left the old socket live out of politeness. A year passes. Then they remove it, and your checkout page stops taking payments on a Tuesday afternoon.

That is the entire problem. It is not exotic. It happens constantly, at every company, with every vendor, and the only defence most teams have is that somebody happens to notice in time.

> **WHY NOBODY FIXES IT**
>
> Updating a vendor connection is never urgent until it is an emergency. It ships no new feature, wins no customer, and appears on no roadmap. It is the software equivalent of servicing the boiler — invisible when done, catastrophic when skipped. So it slips, quarter after quarter, until something breaks or a deadline is forced on the company from outside.

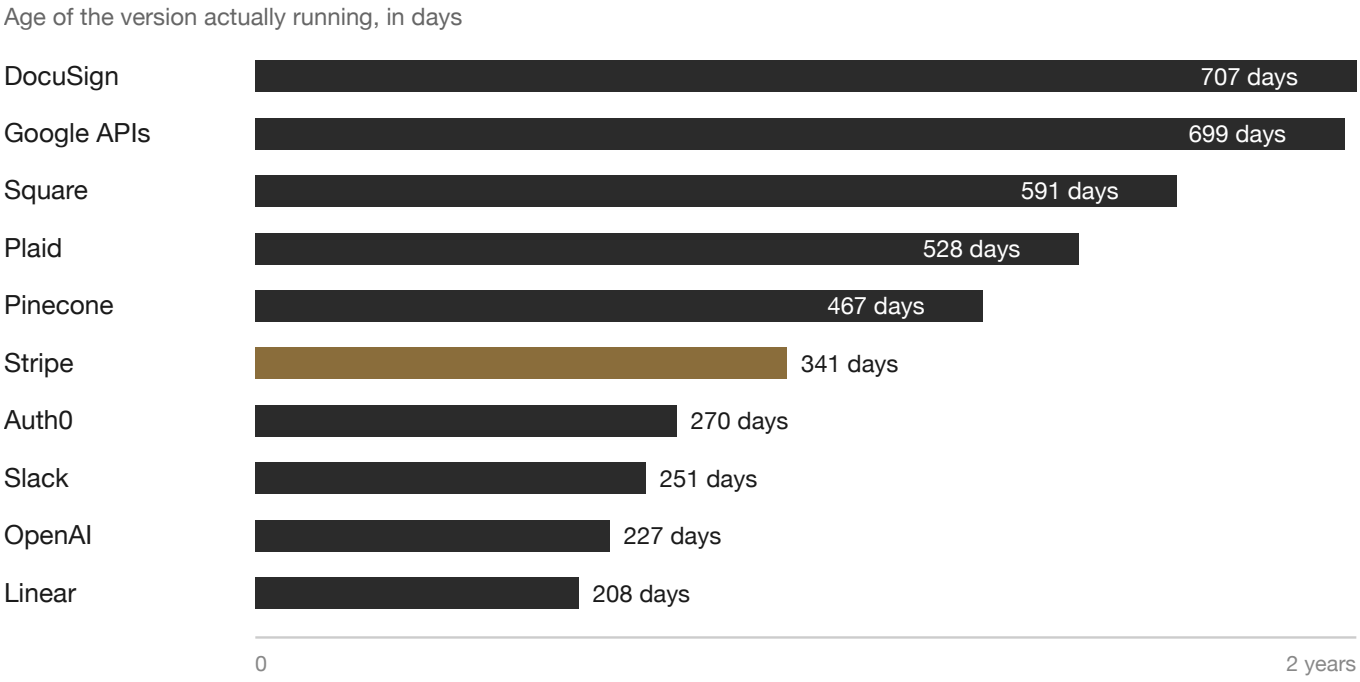
Finding 1 — Customers are running old code

For each of 32 vendors, this study measured every download of their software over one week, grouped by which version was requested, and how long ago that version was published. The result is a picture of what the world is actually running — not what vendors wish it was running.



Ten of the 32 vendors measured. Stripe — one of the most engineering-mature vendors in the sample — is included for contrast, and still sits at 72%.

The second question is how *old* that outdated code is. A month behind is housekeeping. Two years behind is a liability.



Median age of the version in active use. The slowest tenth of users is far worse: for DocuSign and Stripe, that tail runs past four and a half years.

Not every vendor is in the same position. A handful — Supabase, Temporal, PostHog, Resend — have most of their users on current versions. These tend to be younger companies with smaller, more technical customer bases who upgrade willingly. The pattern is consistent: **the older and larger a vendor's customer base, the further behind it falls.** Success creates the problem.

Finding 2 — The ground keeps moving

Falling behind would not matter if vendor software never changed. It changes constantly. Reviewing the full public release history of the same 32 vendors, these are the ones that most often ship changes explicitly described as breaking existing customer code:

VENDOR	BREAKING RELEASES PER YEAR	SHARE OF ALL THEIR RELEASES
Linear	**17.3**	**34%**
Sentry	13.2	12%
Twilio	**9.8**	**42%**
Shopify	6.1	8%
Stripe	5.1	6%
Temporal	4.4	17%
GitHub	3.1	5%
Slack	3.0	8%

VENDOR	BREAKING RELEASES PER YEAR	SHARE OF ALL THEIR RELEASES

Median across all vendors measured: 1.3 breaking releases per year

One vendor breaking something once a year is nothing. But a mid-sized company does not have one vendor. It has thirty. Thirty vendors at the median rate means roughly **forty separate occasions each year** when something a company depends on changes underneath it. Each one is a small decision: notice it or don't, fix it now or later, find out from the changelog or from a customer complaint.

Most of the time, nobody notices. That is what the one-year gap in Finding 1 is made of.

Finding 3 — The damage is visible in public

Engineers complain in public, in writing, with dates attached. A search of public code repositories surfaced **1,234 discussions** from 767 different engineers about a vendor change that broke something or forced an unplanned migration. **31% were still open** — unresolved at the time of writing.

A representative example: one vendor shipped what was labelled a routine maintenance release — the kind teams apply without thinking, because the version number signals "nothing important changed." It broke the basic setup step for every customer who took it. The discussion thread ran for weeks.

This is the texture of the problem. Not dramatic, well-publicised outages, but a steady drip of small failures that each cost a team a day or a week, and never get counted anywhere.

What this research does not yet show

Stated plainly, because a brief that only argues one side is not worth reading:

- **The measurement counts machines as well as people.** Download figures include automated build systems, which repeatedly re-fetch old pinned versions. This inflates the "behind" figure by an unknown amount. The gap is real; its exact size is not yet pinned down.
- **One programming ecosystem was measured.** The 32 vendors were measured through their JavaScript distribution, the only one publishing usable per-version data. Other ecosystems are likely similar but were not verified.

Appendix — How this was measured

Everything in this brief comes from three public sources. No proprietary data, no vendor cooperation, no surveys. Anyone with a laptop can reproduce it. Collected 3–7 August 2026.

The sample: 32 vendors

Algolia · Anthropic · Auth0 · AWS · Clerk · Contentful · Datadog · DocuSign · Firebase · GitHub · Google APIs · HubSpot · LangChain · Linear · MongoDB · Mux · Notion · OpenAI · Pinecone · Plaid · PostHog · Resend · SendGrid · Segment · Sentry · Shopify · Slack · Square · Stripe · Supabase · Temporal · Twilio

Chosen to span the categories a normal company actually buys — payments, messaging, identity, storage, analytics, AI, e-commerce, documents — and to mix decade-old incumbents with companies founded in the last five years. Every vendor in the list returned usable data; none were dropped for producing inconvenient results.

Source 1 — What version is the world running?

Every time a company's build system installs a piece of vendor software, the vendor's distribution network records it, and that record is public. Two public feeds were combined for each vendor:

- **Usage:** the count of installations over the seven days ending 7 August 2026, broken out by which exact version was requested.
- **Publication history:** the date every version of that software was first released.

Joining the two answers "how old is the software people are installing today." The measurement covers **355.7 million installations** spread across **7,190 distinct versions**.

Definitions used, stated so they can be argued with:

- **"Outdated"** means the installation requested a version whose headline number is lower than the vendor's current one. Vendors change that headline number specifically to signal "this release breaks things" — so it is the vendor's own definition of a disruptive change, not one imposed by this research.
- **One exception, applied to one vendor.** Software still in early numbering (versions starting "0.") signals breaking changes in the second number rather than the first. Anthropic was the only vendor in the sample in this state, and was measured on that second number instead. Measuring it the other way would have recorded 100% of its users as current, which would have been false.
- **Age** is the gap between the collection date and the publication date of the version requested. Reported figures are weighted by installation count, so a version installed a million times counts a million times more than one installed twice.
- **Test and preview versions were excluded** entirely — they are not what businesses run.

The distribution channel measured is JavaScript, chosen because it is the only major one that publishes installation counts per version. Every vendor listed also ships versions for other programming languages, which were not measured.

Source 2 — How often do vendors break things?

Every vendor in the sample publishes release notes: a written announcement of what each new version changes. **6,205 releases were collected** across the 32 vendors, spanning October 2014 to August 2026 — the 300 most recent from each.

Each release announcement was scanned for the language vendors use when warning that customer code will stop working: *breaking change*, *backwards incompatible*, *has been removed*, *no longer supported*, *action required*, *migration guide*, *upgrade guide*, *you must update*, and five related phrases. **731 releases carried at least one**. The annual rate for each vendor is that count divided by the time span its releases cover.

Two limits, both material:

- **Five vendors were excluded from the rate.** Google APIs, Clerk, Supabase, LangChain and Datadog publish a separate release for each internal component, so 300 releases cover only weeks rather than years and any per-year rate computed from them is meaningless. The reported median covers the 27 vendors measurable cleanly. Those five still appear in the version-usage findings, which are unaffected.
- **Matching on language over-counts.** Release notes sometimes say "breaking" about something trivial, or link a migration guide that most customers can ignore. The per-vendor figures in Finding 2 are therefore an upper bound on genuinely disruptive releases, not a verified count. Reading a sample by hand to establish the true rate is outstanding work.

Source 3 — Is anyone actually hurt by it?

When a vendor change breaks something, engineers write about it in public, in the discussion threads attached to open code repositories — dated, attributed and searchable.

38 searches were run: one phrase per vendor naming that vendor's changes, plus four general ones ("broke production", and similar). Results were restricted to discussions opened since January 2024, capped at the 40 most-engaged per search, and de-duplicated. That produced **1,234 distinct discussions from 767 different engineers**, dated January 2024 to August 2026, of which **377 (31%) were still unresolved**.

Ranking by engagement pulls in popular repositories alongside genuinely painful migrations. A second pass kept only vendor-specific results whose text contained at least two independent distress markers — *broke*, *deprecated*, *migration*, *upgrade*, *no longer*, *removed*, *stopped working* and related. That narrower, higher-confidence set is **144 discussions from 119 engineers**, and is the set quoted from in this brief.

This is a sample, not a census. It covers only public repositories; the equivalent conversations inside private company code are invisible and certainly far more numerous. Treat 1,234 as a floor with no ceiling attached.

What would strengthen this

- Separating human installations from automated build systems, to establish how much of the 79% is genuine staleness rather than machines re-fetching pinned versions.
- Hand-verifying a sample of the 731 flagged releases to convert an upper bound into a measured rate.
- Repeating the version measurement in a second programming language ecosystem to confirm the pattern holds outside JavaScript.
- Comparing these figures against a vendor's own internal count of how many customers are current — the one number that would validate the whole method.

Figures are point-in-time measurements taken between 3 and 7 August 2026 and will drift as vendors publish. Vendor names appear as measured subjects; none were contacted for, or participated in, this research, and none reviewed these findings before publication.